PROJECTO DE HARDWARE DIGITAL ORIENTADO POR OBJECTOS

João Miguel Fernandes e Ricardo Jorge Machado

Sumário

Os limites entre os domínios do software e do hardware são cada vez mais ténues, pelo que técnicas inicialmente experimentadas no software têm vindo a ser gradualmente aplicadas no hardware. Este artigo pretende descrever o estado actual da utilização da tecnologia de programação orientada por objectos no projecto de hardware digital.

São analisadas as vantagens e implicações quando se introduzem conceitos ligados à tecnologia orientada por objectos em projectos de hardware e é apresentado um exemplo utilizando uma das extensões orientadas por objectos da linguagem VHDL.

Abstract

The boundaries between the software and hardware domains are no longer fixed, which enables the use, on the hardware domain, of techniques originally applied on software projects. This article aims to describe the state-of-the-art in the application of object-oriented programming techniques to digital hardware design.

The advantages and implications of object-oriented technology concepts applied to hardware design are analyzed and an example with an object-oriented VHDL language extension is presented.

1. Introdução

A transferência de técnicas entre os domínios do *software* e do *hardware* tem-se tornado comum, diluindo, ainda mais, as fronteiras entre os dois domínios. A comprová-lo está o facto de o co-projecto de *hardware/software* (*co-design* na nomenclatura inglesa) se ter tornado numa das disciplinas que mais interesse tem despertado em diversas equipas de investigação [1].

No *software*, é agora comum o uso de conceitos tradicionalmente pensados no *hardware*, como, por exemplo, *módulo*, *componente* e *circuito integrado*. O crescente uso de linguagens de programação para a descrição de *hardware* (HDL - *Hardware Description Language*), nomeadamente o VHDL como um *standard* IEEE [2], tornou as funções de um engenheiro de *hardware* bastante semelhantes às de um engenheiro de *software*. Neste sentido, a introdução de tecnologias orientadas por objectos no domínio do *hardware* tem sido ultimamente experimentada com resultados positivos.

À primeira vista, pode parecer forçado utilizar técnicas *orientadas por objectos* (OO) em projectos de *hardware*. Contudo, a prática tem mostrado que o paradigma orientado por objectos é aceite com relativa naturalidade pelos engenheiros de *hardware*, já que estes se habituaram a lidar com objectos (placas, componentes, módulos, células) que comunicam por sinais [3].

As abordagens orientadas por objectos no desenvolvimento de *software* conheceram grande aceitação por permitirem gerir com relativa facilidade a complexidade dos sistemas e por aumentarem significativamente a reutilização [4].

Estes dois conceitos, assim como a possibilidade de descrição a um nível mais elevado de abstracção, uma maior legibilidade e uma manutenção facilitada dos projectos, são os principais responsáveis pelas extensões que têm sido sugeridas para adicionar conceitos orientados por objectos à linguagem VHDL [5-10].

Neste artigo, são descritos os conceitos orientados por objectos na perspectiva da sua aplicação em projectos de *hardware* digital e é apresentado um exemplo da aplicação destes conceitos utilizando uma extensão do VHDL.

Na escrita deste artigo assumiu-se que o leitor domina os fundamentos dos sistemas digitais e que possui conhecimentos básicos de VHDL e de programação orientada por objectos.

2. Conceitos OO Aplicados ao Hardware

A aplicação da modelação orientada por objectos ao domínio do *hardware* exige a re-interpretação de conceitos originalmente desenvolvidos para aplicar ao domínio do *software*. Os conceitos que são necessários re-analisar são os seguintes:

- Abstracção de Dados;
- Hierarquia e Herança;
- Mensagens e Polimorfismo;
- Encapsulamento e Modularidade.

2.1 Abstracção de Dados

A *abstracção de dados* significa que as estruturas de dados são definidas em conjunto com as operações que as utilizam. Ao conjunto definido pelas estruturas de dados e pelas operações dá-se o nome de

classe. Em terminologia orientada por objectos, às estruturas de dados e às operações que constituem uma classe chama-se, respectivamente, *variáveis de instância* e *métodos*. Um *objecto* é uma instância de uma classe, contém espaço próprio para as suas variáveis de instância e o acesso a estas é unicamente possível usando os métodos definidos para a respectiva classe. Todos os objectos de uma classe usam os mesmos métodos que, na implementação, são gerados uma só vez.
A abstracção de dados é muito natural em *hardware*. Por exemplo, o conteúdo de um contador é apenas acessível através de operações conhecidas (fig. 1). Um acesso directo ao conteúdo do contador não é usualmente permitido.

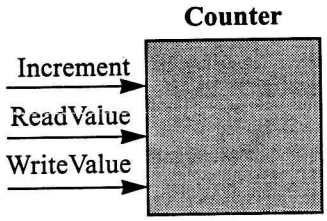

Fig. 1: Classe "Counter".

2.2 Hierarquia e Herança

A criação de variantes de classes sem utilizar técnicas orientadas por objectos pode ser conseguida por "cópia e modificação". Daqui resultam inúmeras versões que diferem ligeiramente, contêm redundância e cuja manutenção se torna progressivamente dificultada. A coerência entre classes pode ser mantida, mas o "preço" a pagar é muito alto: as modificações têm de realizar-se em muitas partes.
Com uma hierarquia de classes e um mecanismo de herança garante-se uma maior reutilização das classes, devido a uma gestão mais facilitada.
A *herança* é definida como o mecanismo de construir novas classes a partir de uma outra classe (*superclasse*). Uma *subclasse* herda as variáveis de instância e os métodos da sua superclasse. As subclasses podem adicionar novas estruturas de dados e novos métodos àqueles já existentes nas superclasses, bem como redefinir métodos destas. A herança entre classes permite a especialização de componentes. Por exemplo, uma possível especialização *AddCounter* da classe *Counter* da fig. 1 pode obter-se acrescentando um novo método *AddValue*.
O uso de herança múltipla (uma subclasse ter mais do que uma superclasse) mostra-se igualmente útil em determinadas situações. Considere-se, por exemplo, duas classes *LeftShiftRegister* e *RightShiftRegister*, que sendo usadas como superclasses permitiriam obter uma classe *BidirectionalShiftRegister* que suporta ambos os deslocamentos, sem a necessidade de escrever nenhum método novo.

2.3 Mensagens e Polimorfismo

Por vezes, numa subclasse, é necessário redefinir o comportamento de um método definido numa superclasse. O processo de redefinir o comportamento de um método herdado chama-se *reescrita* de métodos. Desta forma, o método reescrito tem o mesmo nome do herdado, mas comporta-se diferentemente conforme o objecto em causa. Este mecanismo de sobreposição implementa um comportamento *polimórfico*.
Por exemplo, em alternativa à definição do método *AddValue* na subclasse *AddCounter*, poder-se-ia reescrever o método *Increment*, de modo a incluir um argumento de entrada que indicasse a quantidade a adicionar ao conteúdo do contador.
A forma dos objectos comunicarem entre si, representando no mundo do *hardware* os sinais e os protocolos entre componentes, consiste no envio de mensagens que efectuam a invocação de métodos aos objectos receptores. Uma mensagem indica apenas a operação a executar, deixando a responsabilidade de como o fazer para o objecto receptor da mensagem.

2.4 Encapsulamento e Modularidade

No domínio do *hardware*, o encapsulamento e a modularidade proporcionam os mesmos benefícios no desenvolvimento e manutenção de módulos que aqueles que se obtêm em aplicações de *software*.
O *encapsulamento* permite o uso de funcionalidades sem existir a necessidade de conhecer os pormenores das suas implementações, pelo que alterações nestas não implicam modificações na interface.
O conceito de *modularidade* tem uma enorme importância na implementação de HDLs, uma vez que os componentes de *hardware* constituem as unidades de modularidade, ou seja, a modularidade é intrínseca ao próprio *hardware*.

3. Exemplo: Extensão OO para VHDL

A linguagem VHDL pode ser classificada como uma linguagem baseada, e não orientada, em objectos, o que significa que suporta a funcionalidade dos objectos, mas não a sua gestão. Uma linguagem diz-se orientada por objectos se suporta a funcionalidade dos objectos, a gestão de objectos por classes e a gestão de classes por hierarquia [10].

Em VHDL, o par *Entidade/Arquitectura* pode ser usado para implementar uma classe: a entidade define a interface da classe e a arquitectura descreve uma possível implementação para a classe. Os sinais VHDL podem usar-se para a comunicação entre entidades.
Uma das várias extensões para o VHDL, OO-VHDL [9], que adiciona mecanismos orientados por objectos ao VHDL, oferece uma unidade adicional de modularidade o par *EntidadeObjecto/Arquitectura*, que descreve os objectos e o seu comportamento. A definição de uma EntidadeObjecto permite a utilização dos mecanismo de herança recorrendo a hierarquias de EntidadesObjecto.
Em OO-VHDL, a descrição da classe *Counter* (fig. 1) é efectuada através da declaração da seguinte EntidadeObjecto:

```
EntityObject Counter is
    -- Parte da Entidade
    generic (maxValue:integer);
    port (clock: in bit);
    -- Parte do Objecto
    operation Increment;
    operation ReadValue(OutVal: out integer);
    operation WriteValue(InVal: in integer);
end Counter;

Architecture OOCounter of Counter is
    instance variable Value:integer;
    operation Increment is
    begin
        if (Value>= maxValue) then
            Value:=0;
        else
            Value:=Value+1;
        end if;
    end;
    operation ReadValue(OutVal: out integer) is
    begin
        OutVal:=Value;
    end;
    operation WriteValue(InVal: in integer) is
    begin
        Value:=InVal;
    end;
begin
    -- corpo nulo
end OOCounter;
```

As variáveis e os métodos definidos na classe *Counter* são herdados pela classe *AddCounter*. Daqui resultam evidentes vantagens: reutilização de *software*, partilha de código, comportamentos uniformes e prototipagem rápida facilitada. A definição da classe *AddCounter* é feita como a seguir se mostra:

```
EntityObject AddCounter is new Counter
    -- "is new" declara uma subclasse de Counter
    -- Parte da Entidade herdade de Counter
    -- Parte do Objecto
    operation Add (InVal: in integer);
end AddCounter;

Architecture OOAddCounter of AddCounter is
  operation Add(InVal: in integer) is
  begin
      Value:=Value+InVal;
      if (Value>= maxValue) then
      Value:=Value-maxValue;
      end if;
  end;
end OOAddCounter;
```

O código OO-VHDL pode ser traduzido em VHDL usando um pré-processador, o que permite a utilização de qualquer ferramenta CAD, para simulação ou síntese de *hardware*, que aceite VHDL como formato de entrada.
Da mesma forma, existem outras ferramentas que permitem modelar *hardware* com objectos, gerando VHDL mas baseadas noutros formatos de especificação, que não extensões à linguagem VHDL. Por exemplo, a ferramenta SOFHIA [11] baseia-se em RdP-shobi (um tipo de Redes de Petri Orientadas por Objectos) [12].

4. Conclusões

Os exemplos apresentados neste artigo ilustram a reutilização de código e a facilidade com que novos e mais especializados componentes podem ser criados, com reflexos importantes na diminuição do tempo necessário para modelar o sistema. Desta forma, a introdução do paradigma de modelação orientado por objectos em projectos de *hardware* digital apresenta muitas vantagens, nomeadamente no suporte a um alto nível de reutilização através da manutenção e da modificação de modelos, na facilidade de instanciação de componentes com diferentes parâmetros e na possibilidade de recorrer a técnicas de *software* para verificação e síntese dos modelos. Porém, o recurso a esta nova abordagem deve ser bem equacionado e não visto como uma panaceia para problemas em qualquer área de aplicação. Por exemplo, a criação e remoção dinâmica de objectos, tão útil em projectos de *software*, terá de ser usada com cuidado em projectos de *hardware*, apesar de já ser possível vislumbrar algumas uti-

lizações em projectos que incluam novas famílias de FPGAs e CPLDs.
Esta abordagem, conjuntamente com as novas extensões de VHDL, irá contribuir para um maior desenvolvimento da classe de problemas abordada pelo co-projecto *hardware/software*.

5. Referências

[1] De Micheli, G. *Computer-Aided Hardware/ /Software Co-Design*. IEEE Micro Magazine, Agosto 1994.

[2] IEEE. *IEEE Standard VHDL Language Reference Manual*. New York, USA, 1994.

[3] Kumar, S., J.H. Aylor, B.W. Johnson, Wm.A. Wulf. *Object-Oriented Techniques in Hardware Design*. IEEE Computer, pp.64-70, Junho 1994.

[4] Budd, T. *An Introduction to Object-Oriented Programming*. Addison-Wesley Publishing Company, 1991.

[5] Glunz, W. *Extensions from VHDL to VHDL++*. Technical Report JESSI-AC/S2-WP1-T2.4-Q3, Siemens AG, 1991.

[6] Dunlop, D. D. *Object-Oriented Extensions to VHDL*. In Proc. VIUF Fall 1994 Conf., pp. 5.1--5.9, 1994.

[7] Schumacher, G.; W. Nebel. *Inheritance Concept for Signals in Object-Oriented Extensions to VHDL*. In Proceedings of EURO--DAC with EURO-VHDL'95, pp. 428-35, Setembro 1995.

[8] Agsteiner, K.; D. Monjau; S. Schulze. *Object-Oriented High-Level Modelling of System Components for the Generation of VHDL*. In Proceedings of EURO-DAC with EURO-VHDL'95, pp. 436-41, Setembro 1995.

[9] Swamy S.; A. Molin; B.M. Covnot. *OO-VHDL: Object-Oriented Extensions to VHDL*. IEEE Computer, pp. 18-26, Outubro 1995.

[10] Cabanis, D.; S. Medhat; N. Weavers. *Object--Oriented Extensions to VHDL: The Classification Orientation*. In VHDL User Forum Europe - SIG-VHDL Spring'96 Working Conf., pp. 9-20, Maio 1996.

[11] Machado, R.J.; J.M. Fernandes; Á.J. Proença. *SOFHIA: A CAD Environment to Design Digital Control Systems*. In XIII IFIP Conference on Computer Hardware Description Languages and Their Applications - CHDL'97, Toledo, Espanha, Chapman & Hall, Abril 1997.

[12] Machado R.J., J.M. Fernandes, A.J. Proença. *Specification of Industrial Digital Controllers with Object-Oriented Petri Nets*. In IEEE International Symposium on Industrial Electronics - ISIE'97, Guimarães, Portugal, Julho 1997.

Informação sobre os autores

João Miguel Lobo Fernandes é Licenciado em Engenharia de Sistemas e Informática (1991) e Mestre em Informática (1994) pela U. Minho. Foi Investigador da U. Bristol-UK. Está a preparar o Doutoramento em Engenharia de Computadores no Dep. de Informática da U. Minho, onde é Assistente. Desenvolve investigação na área dos Sistemas Digitais, onde tem várias publicações em congressos e revistas internacionais.

Ricardo Jorge Silvério de Magalhães Machado é Licenciado em Engenharia Electrotécnica e de Computadores (1994) pela FEUP e Mestre em Informática (1996) pela U. Minho. Foi Investigador do INESC-Porto e da ENSEA-Paris. Como Engenheiro da Texas Instruments, registou, nos EUA, a patente dum sistema electrónico. Investiga na área do co-projecto hw/sw e está a preparar o Doutoramento em Engenharia de Computadores no Dep. de Informática da U. Minho, onde é Assistente.